Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

Data Structures and Algorithms – C++ Implementation

Huỳnh Tấn Đạt

Email: htdat@cse.hcmut.edu.vn

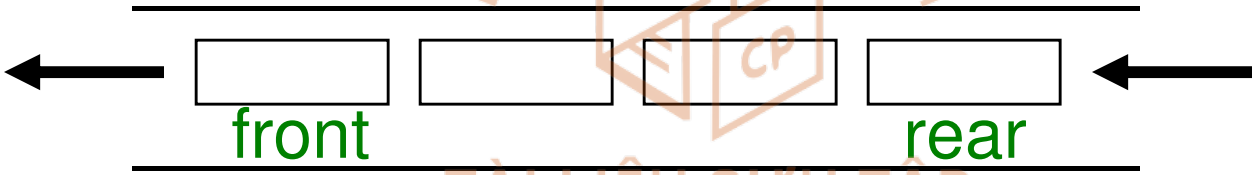
Home Page: http://www.cse.hcmut.edu.vn/~htdat/

Queues

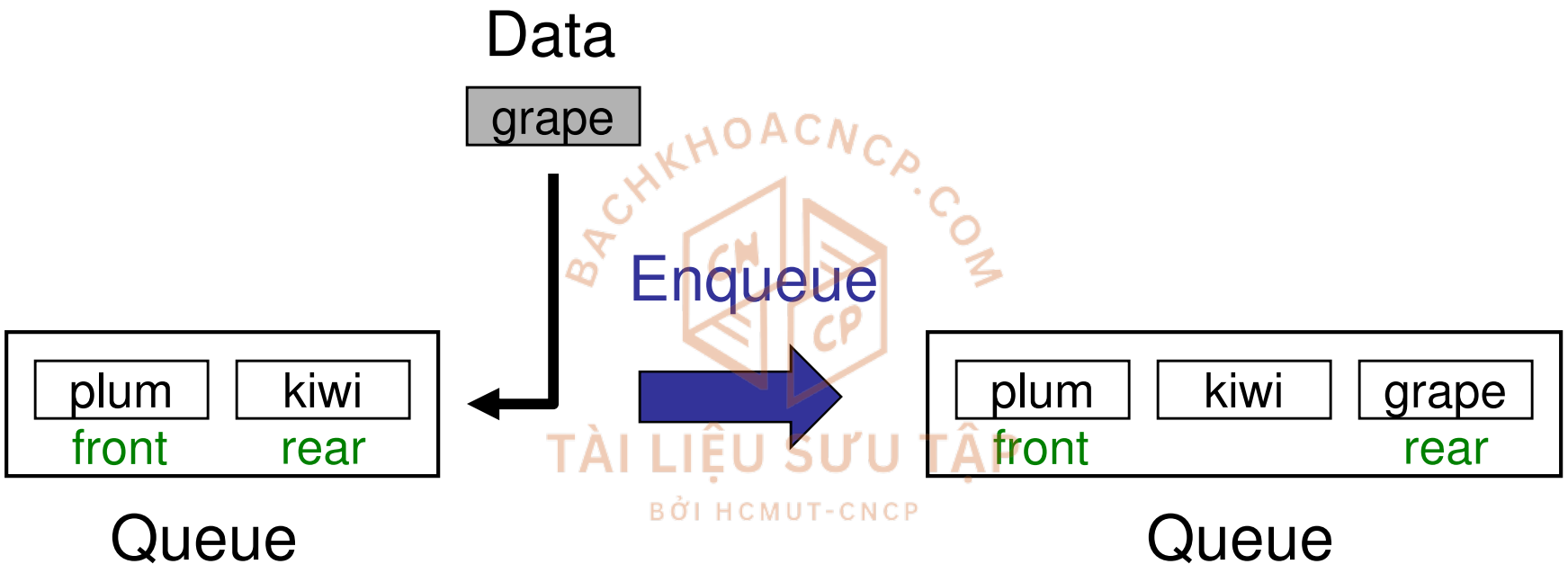
- Basic queue operations
- Linked-list implementation
- Queue applications
- Array implementation

Queues

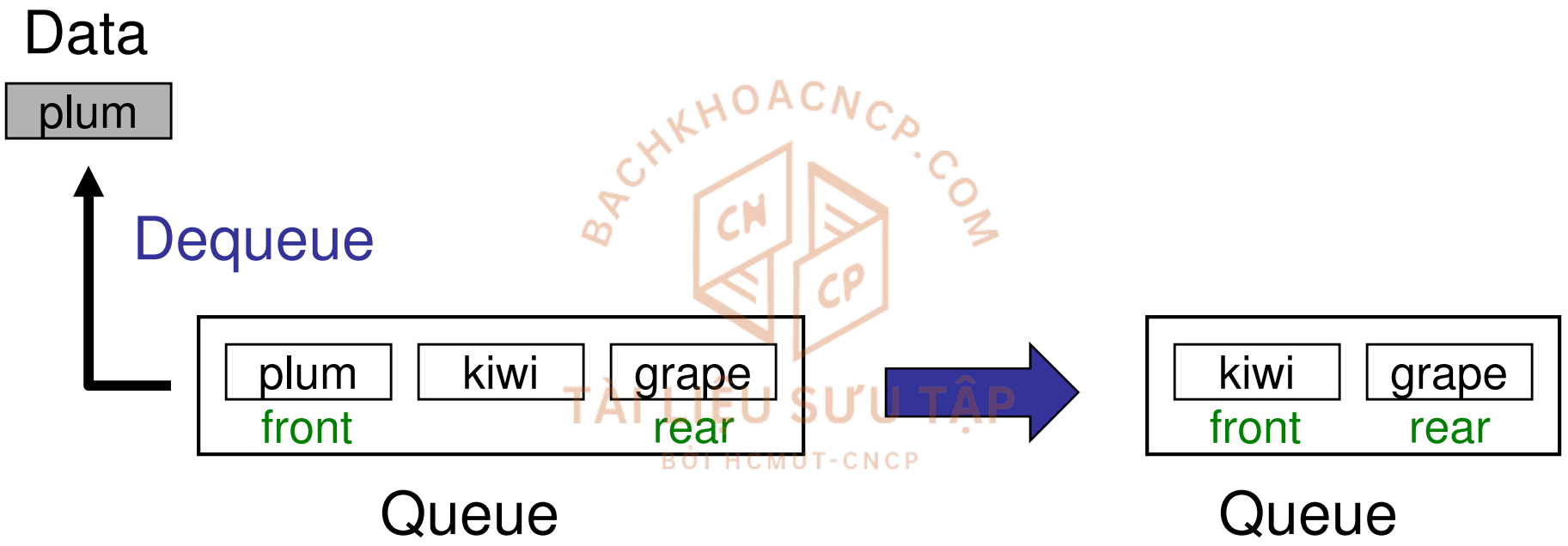
- Data can only be inserted at one end called the rear, and deleted from the other end called the front.
- First-In-First-Out (FIFO) data structure.



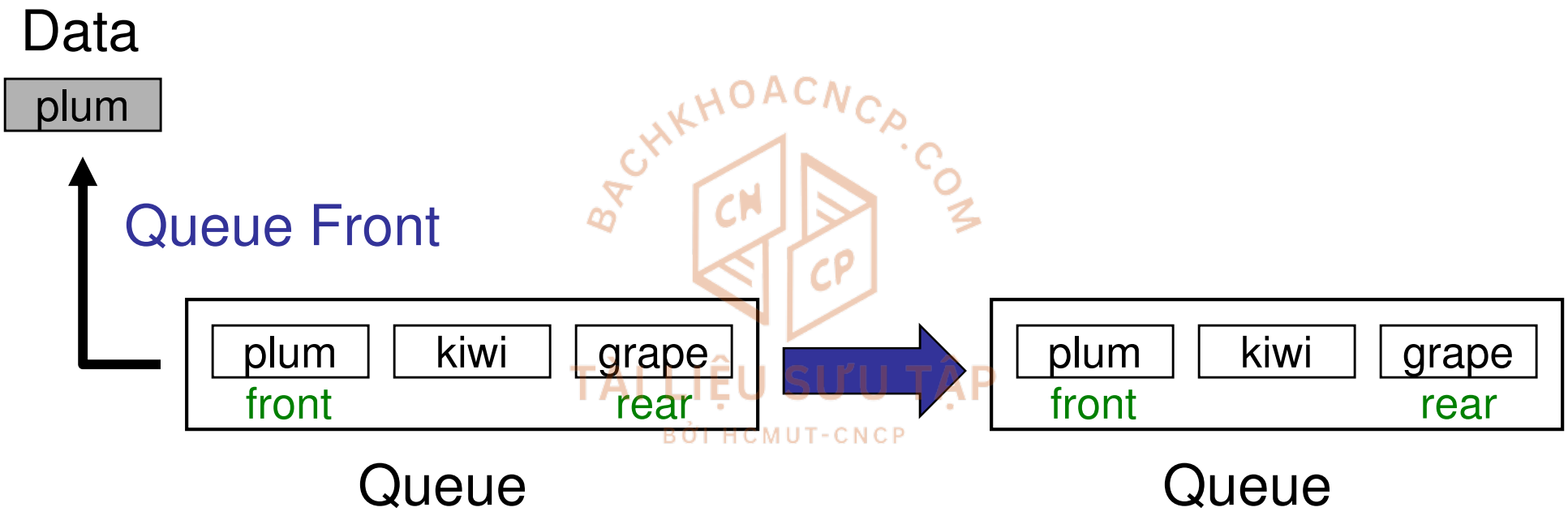
Basic Queue Operations



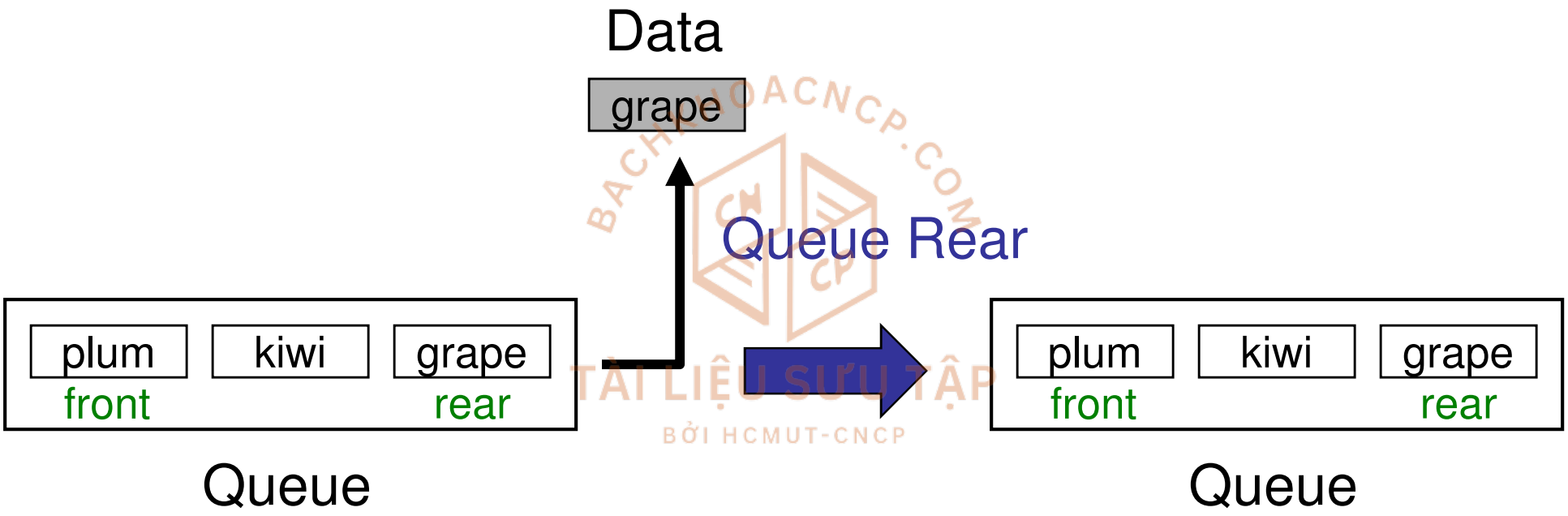
Basic Queue Operations



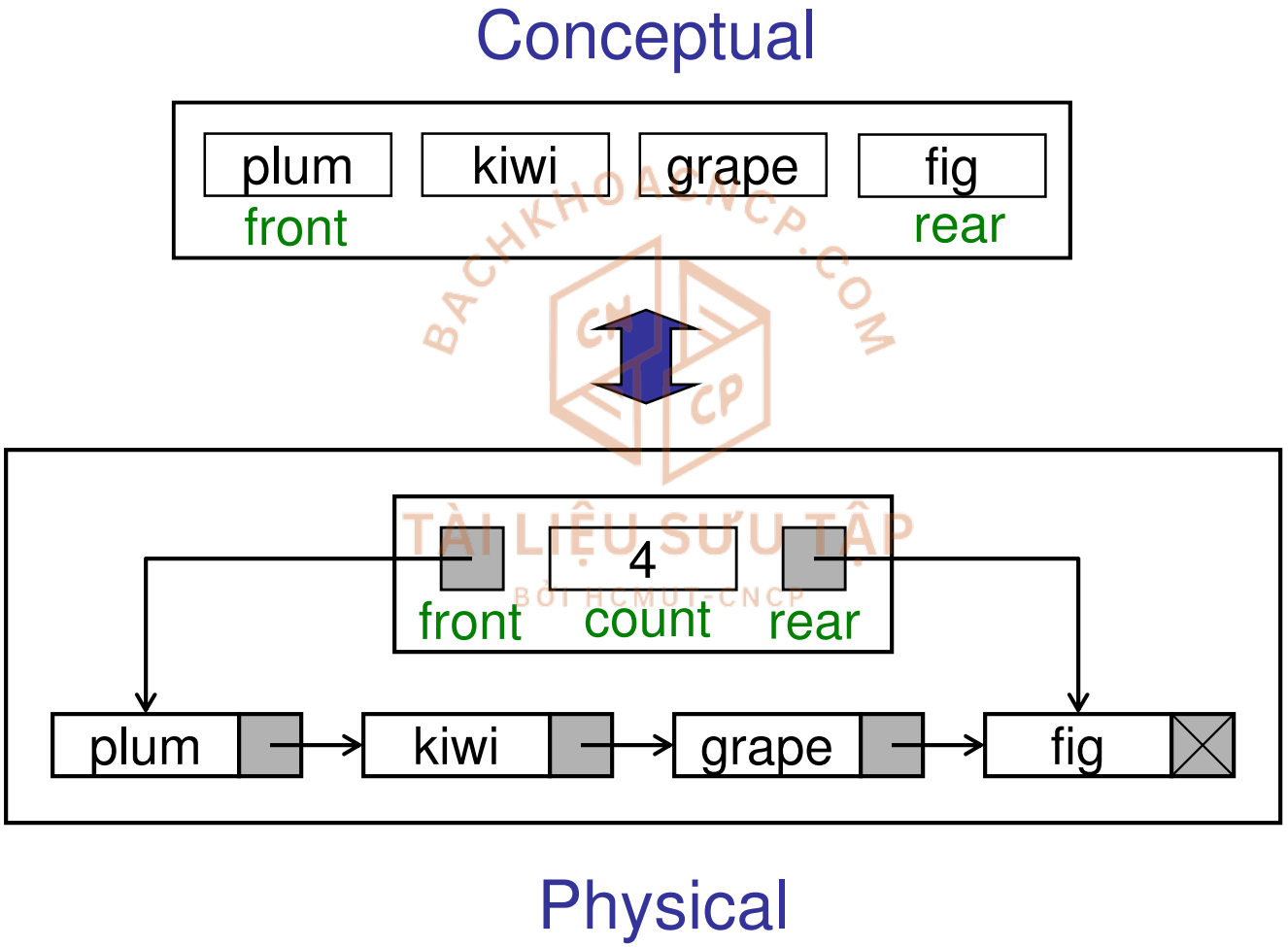
Basic Queue Operations



Basic Queue Operations

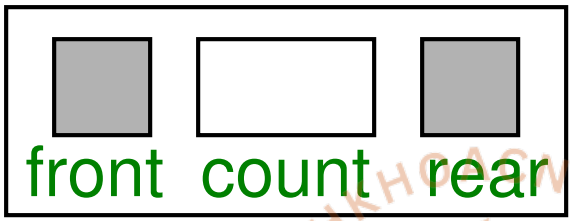


Basic Queue Operations



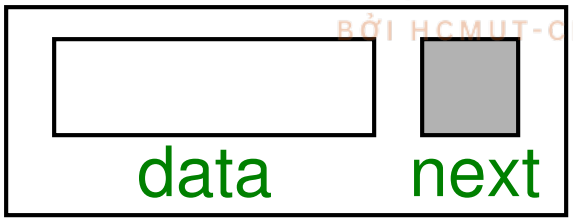
Linked-List Implementation

Queue structure



queue
    front <node pointer>
    count <integer>
    rear <node pointer>
end queue

Queue node structure



node
    data <dataType>
    next <node pointer>
end node

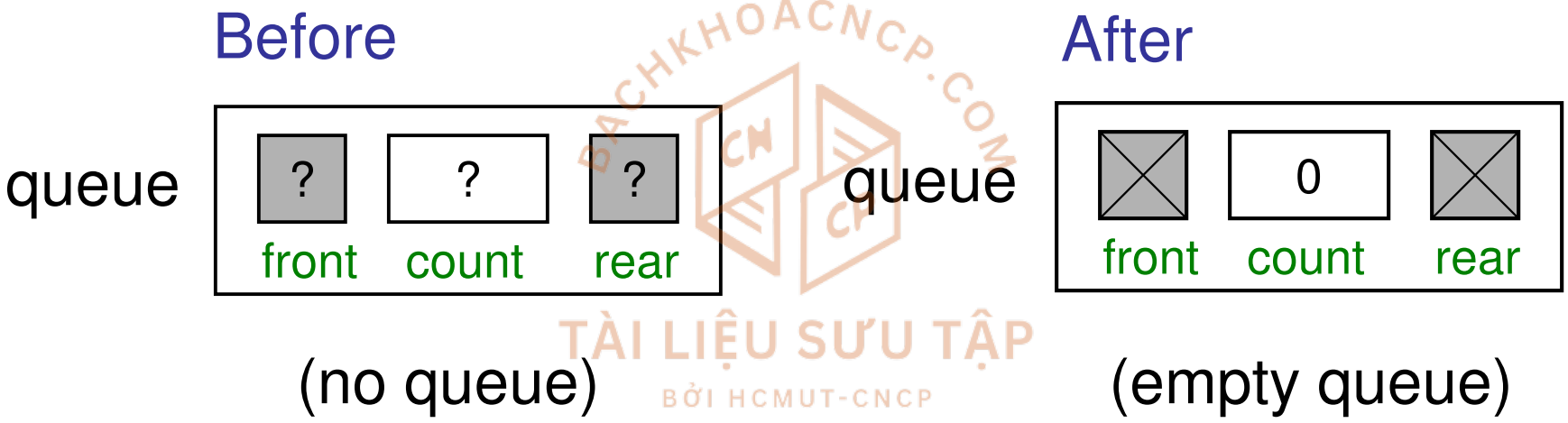
Linked-List Implementation

```
template <class ItemType>
struct Node {
  ItemType data;
  Node<ItemType> *next;
};

template <class List_ItemType>
class Queue{
public:
  Queue();
  ~Queue();
```

Linked-List Implementation

```
  void Enqueue(List_ItemType dataIn);
  int Dequeue(List_ItemType &dataOut);
  int GetQueueFront(List_ItemType &dataOut);
  int GetQueueRear(List_ItemType &dataOut);
  void Clear();
  int IsEmpty();
  int GetSize();
  Queue<List_ItemType>* Clone();

private:
  Node<List_ItemType>* front, *rear;
  int count;
};
```

Create Queue



Create Queue

Algorithm createQueue (ref queue <metadata>

Initializes the metadata of a queue

Pre queue is a metadata structure of a queue

Post metadata have been initialized

1 queue.front = null

2 queue.rear = null

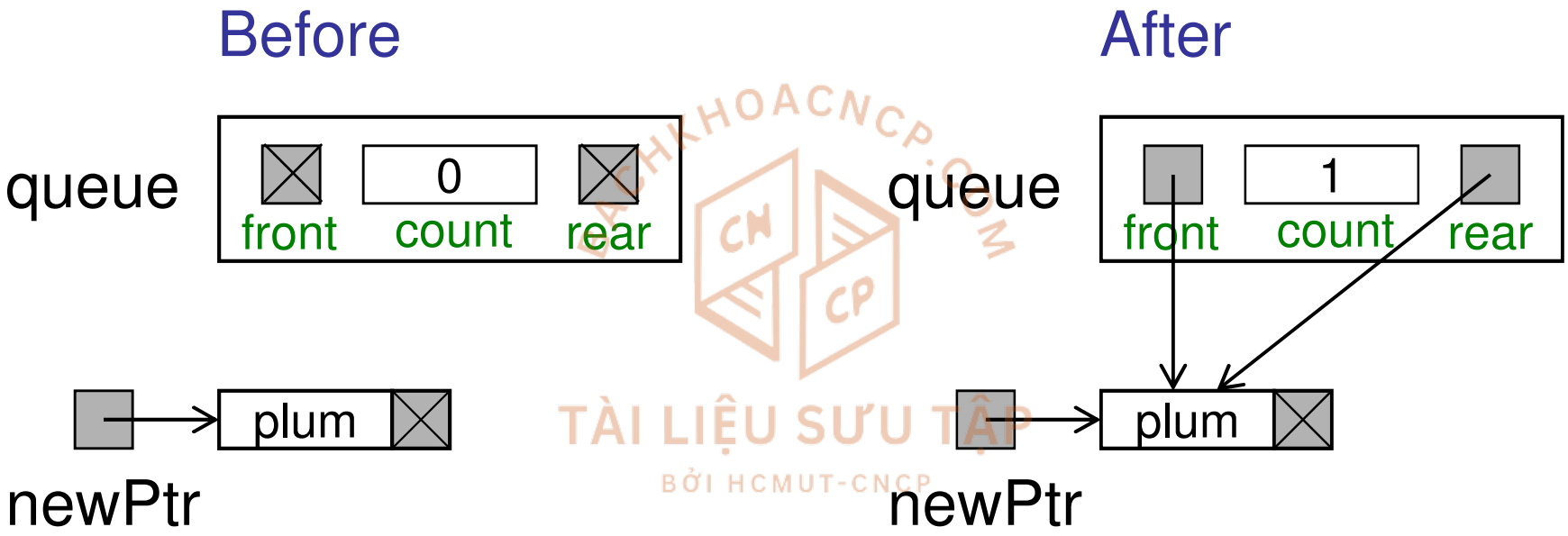
3 queue.count = 0

End createQueue

Create Queue

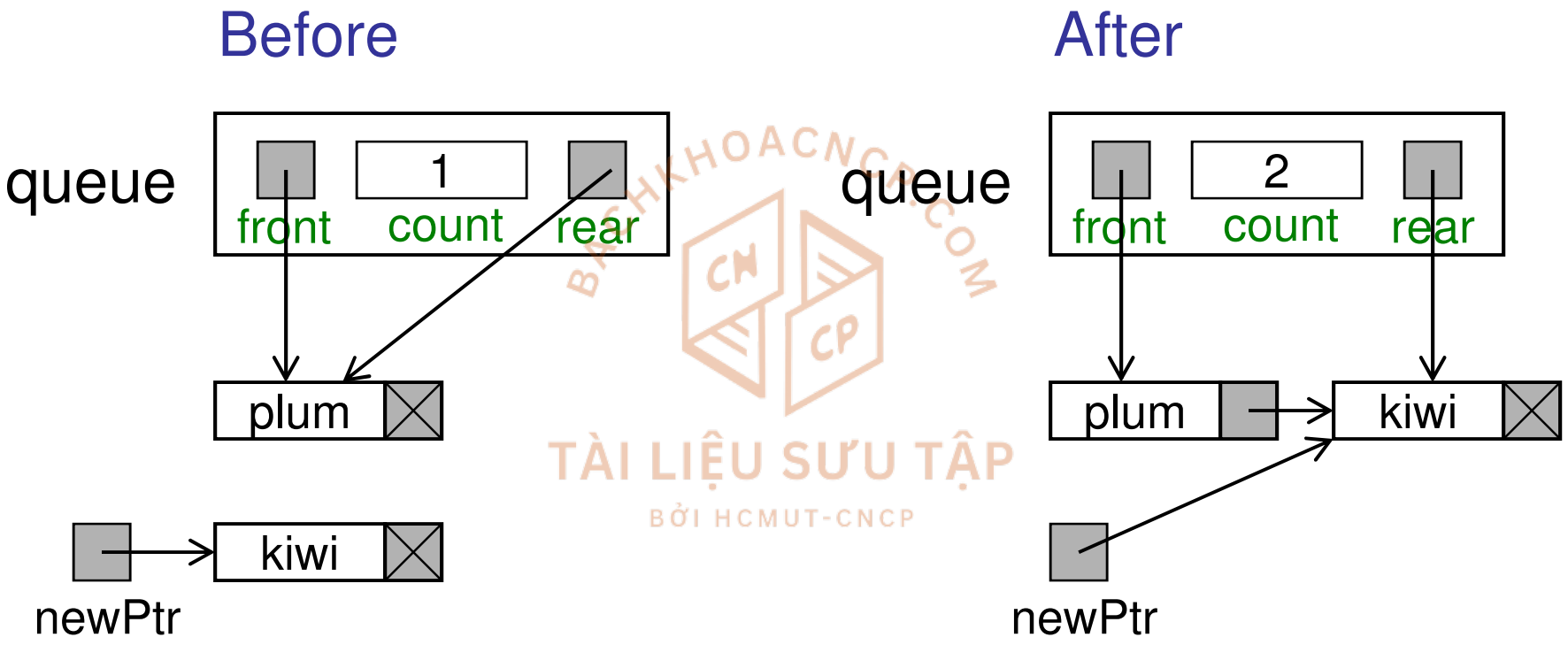
```
template <class List_ItemType>
Queue<List_ItemType>::Queue(){
  this->front = this->rear = NULL;
  this->count = 0;
}

template <class List_ItemType>
Queue<List_ItemType>::~Queue(){
  this->Clear();
}
```

Enqueue



Insert into null queue

Enqueue



Insert into queue with data

Enqueue

Algorithm enqueue (ref queue <metadata>,
val data <dataType>)

Inserts one item at the rear of the queue

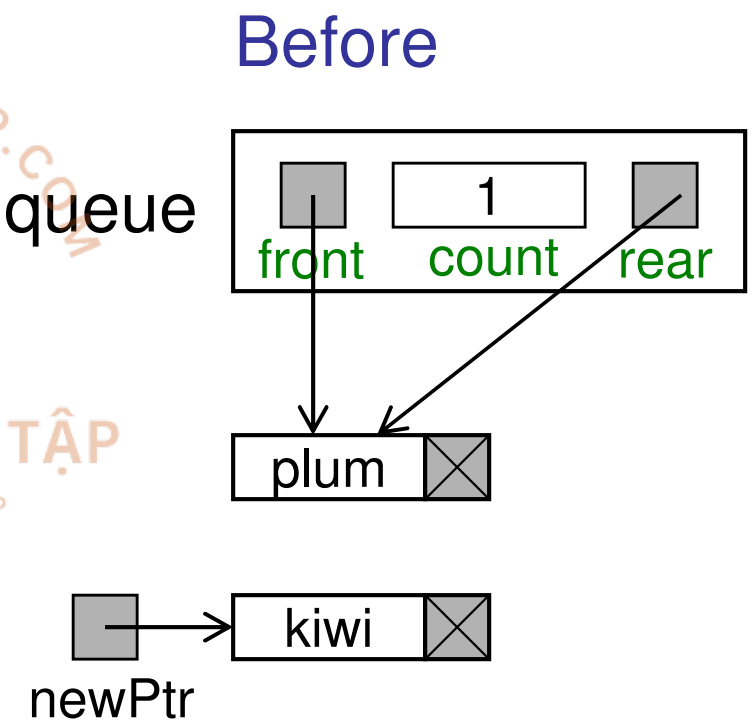
Pre queue is a metadata structure of a valid queue
data contains data to be inserted into queue

Post data have been inserted in queue

Return true if successful, false if memory overflow

Enqueue

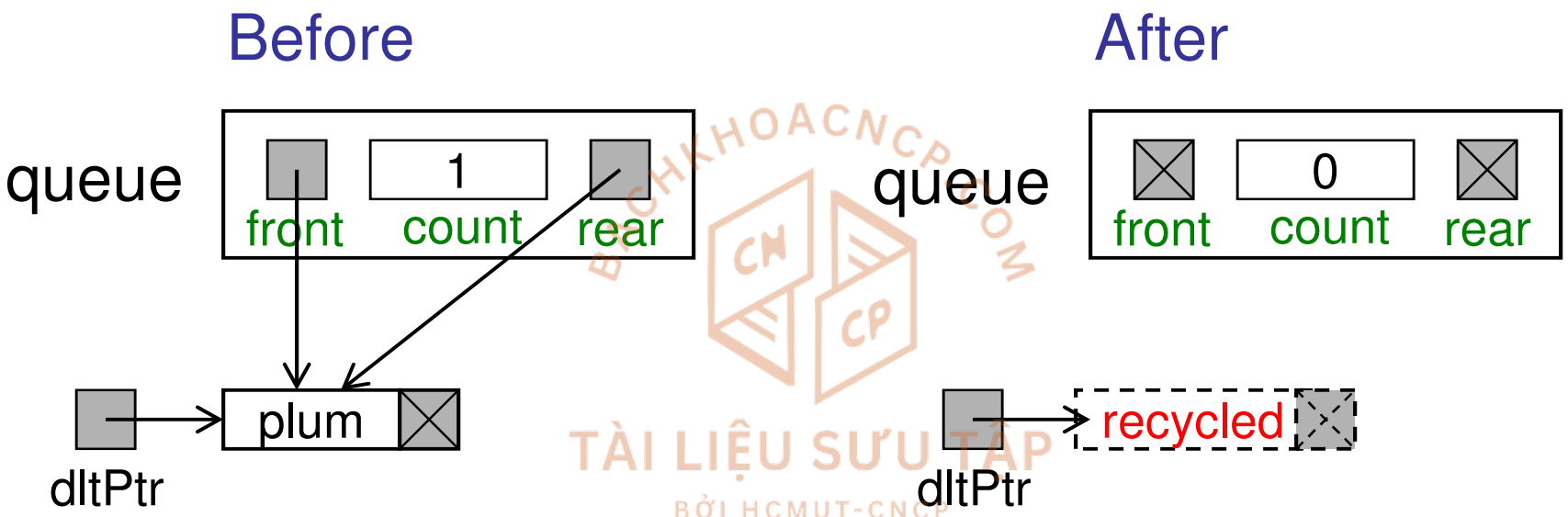
```
1   if (queue full)
    1   return false
2   allocate (newPtr)
3   newPtr -> data = data
4   newPtr -> next  = null
5   if (queue.count = 0)
        Insert into null queue
    1   queue.front = newPtr
6   else
        Insert into queue with data
    1   queue.rear -> next = newPtr
7   queue.rear = newPtr
8   queue.count = queue.count + 1
9   return true
End  enqueue
```



Enqueue

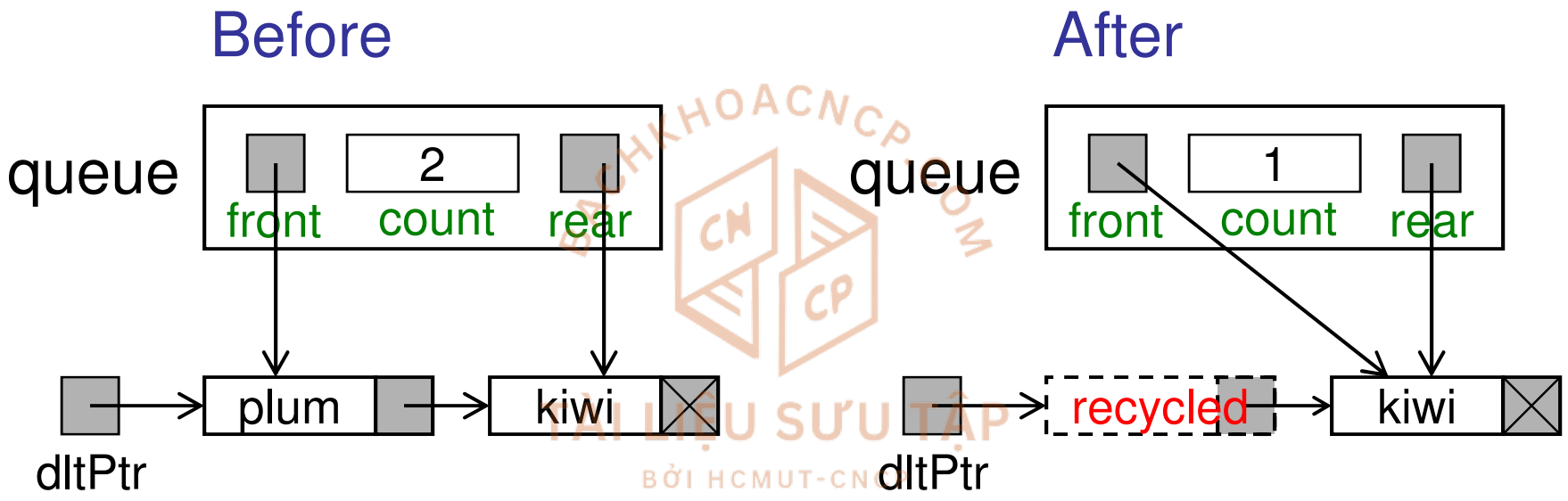
```
template <class List_ItemType>
void Queue<List_ItemType>::Enqueue
  (List_ItemType value){
  Node<List_ItemType>* newPtr = new
                        Node<List_ItemType>();
  newPtr->data = value;
  newPtr->next = NULL;
  if (this->count == 0)
     this->front = newPtr;
  else
     this->rear->next = newPtr;
  this->rear = newPtr;
  this->count++;
}
```

Dequeue



Delete data in queue with only one item

Dequeue



Delete data in queue with more than one item

Dequeue

Algorithm dequeue (ref queue <head pointer>,
ref dataOut <dataType>)

Deletes one item at the front of the queue and returns its data to caller

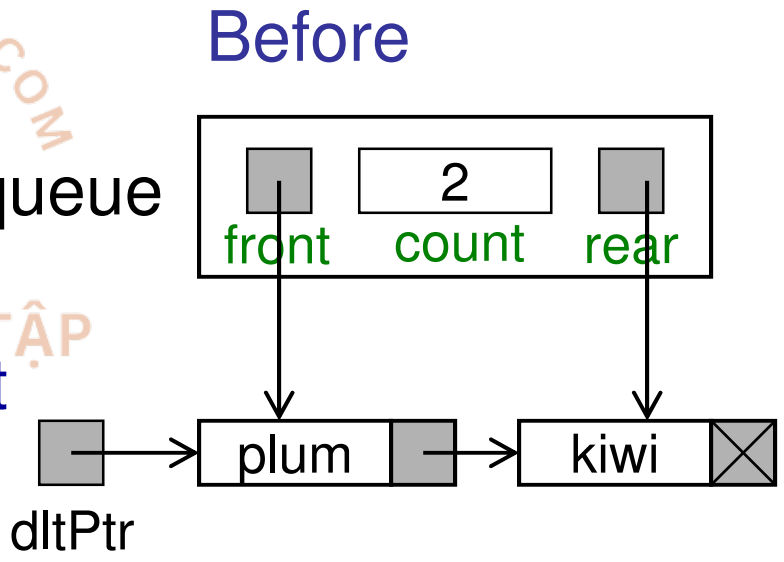
Pre queue is a metadata structure of a valid queue
dataOut is to receive dequeued data

Post front data have been returned to caller

Return true if successful; false if underflow

Dequeue

```
1  if (queue empty)
   1  return false
2  dataOut = queue.front -> data
3  dltPtr = queue.front
4  if (queue.count = 1)
       Delete data in queue with
       only one item
   1  queue.rear = null
5  queue.front = queue.front -> next
6  queue.count = queue.count - 1
7  recycle (dltPtr)
8  return true
End dequeue
```



Dequeue

```
template <class List_ItemType>
int Queue<List_ItemType>::Dequeue(List_ItemType
  &dataOut){
  if (count == 0)
     return 0;
  dataOut = front->data;
  Node<List_ItemType>* dltPtr = this->front;
  if (count == 1)
     this->rear = NULL;
  this->front = this->front->next;
  this->count--;
  delete dltPtr;
  return 1;
}
```

Queue Front & Queue Rear

```
template <class List_ItemType>
int Queue<List_ItemType>::GetQueueFront
   (List_ItemType &dataOut){
   if (count == 0)
      return 0;
   dataOut = this->front->data;
   return 1;
}
template <class List_ItemType>
int Queue<List_ItemType>::GetQueueRear
   (List_ItemType &dataOut){
   if (count == 0)
      return 0;
   dataOut = this->rear->data;
   return 1;
}
```

Destroy Queue

Algorithm destroyQueue (ref queue <metadata>)
Deletes all data from a queue
Pre queue is a metadata structure of a valid queue
Post queue empty and all nodes recycled
Return null pointer

```
1  if (queue not empty)
   1  loop (queue.front not null)
      1  temp = queue.front
      2  queue.front = queue.front -> next
      3  recycle (temp)
2  queue.front = null
3  queue.rear = null
4  queue.count = 0
5  return
```

End destroyQueue

Clear Queue

```
template <class List_ItemType>
void Queue<List_ItemType>::Clear() {
  Node<List_ItemType>* temp;
  while (this->front != NULL){
     temp = this->front;
     this->front= this->front->next;
     delete temp;
  }
  this->front = this->rear = NULL;
  this->count = 0;
}
```

Queue Empty

```
template <class List_ItemType>
int Queue<List_ItemType>::IsEmpty() {
  return (count == 0);
}

template <class List_ItemType>
int Queue<List_ItemType>::GetSize() {
  return count;
}
```

Print Queue

```
template <class List_ItemType>
void Queue<List_ItemType>::Print2Console(){
  Node<List_ItemType>* p;
  p = this->front;
  printf("Front: ");
  while (p != NULL){
     printf("%d\t", p->data);
     p = p->next;
  }
  printf("\n");
}
```

Using Queues

```
int main(int argc, char* argv[]){
  Queue<int> *myQueue = new Queue<int>();
  int val;
  myQueue->Enqueue(7);
  myQueue->Enqueue(9);
  myQueue->Enqueue(10);
  myQueue->Enqueue(8);
  myQueue->Dequeue(val);
  delete myQueue;
  return 1;
}
```

Exercises

```
template <class List_ItemType>
Queue<List_ItemType>*
          Queue<List_ItemType>::Clone() {
  // ...
}
```